BAB 2
LANDASAN TEORI

2.1 Arsitektur Microservices

Arsitektur *microservices* merupakan pendekatan pengembangan perangkat lunak yang memecah satu aplikasi utuh menjadi sekumpulan layanan kecil yang berdiri sendiri. Setiap layanan tersebut berfokus pada satu fungsi bisnis tertentu dan bersifat *loosely coupled*, yaitu keterkaitan antarlayanan yang longgar sehingga perubahan pada satu layanan tidak mengharuskan perubahan pada layanan lain [19]. Pemisahan ini membuat setiap komponen harus beroperasi secara terisolasi, sehingga pertukaran data antarlayanan hanya dapat dilakukan melalui antarmuka yang telah ditentukan sebelumnya [20]. Karakteristik mandiri inilah yang menjadi pembeda utama dari sistem monolitik tradisional, sekaligus membuka jalan bagi proses komputasi yang lebih tersebar.

Sistem yang tersebar seperti ini membutuhkan cara komunikasi yang efisien agar layanan-layanan dapat bekerja sama tanpa bergantung pada satu pengendali pusat. Layanan-layanan tersebut dapat saling bertukar informasi secara langsung dan asinkron tanpa bergantung pada satu pusat kendali [20]. Cara komunikasi yang ringan seperti *Representational State Transfer* (REST), untuk komunikasi antar layanan, dan *Message Queue* (MQ), untuk pengiriman pesan secara asinkron, membantu membagi beban pemrosesan dan penyimpanan data di seluruh infrastruktur [19]. Pembagian beban ini membuat sistem lebih tahan terhadap kerusakan perangkat keras karena kegagalan pada satu bagian tidak langsung melumpuhkan keseluruhan sistem. Di sisi lain, hilangnya kendali terpusat justru mempersulit proses pengawasan dan pelacakan aliran data antarlayanan.

2.1.1 Cascading Failures

Kegagalan berantai (*cascading failure*) terjadi ketika gangguan pada satu layanan menyebar ke layanan lain melalui rantai pemanggilan (*call chain*), sehingga gangguan yang dirasakan pengguna di antarmuka depan sering kali tidak mencerminkan letak kerusakan sebenarnya di lapisan belakang (*backend*) sistem [21]. Diagnosis kegagalan berantai karena itu memerlukan pembedaan antara gejala (*symptom*), yaitu anomali turunan seperti hilangnya respons koneksi atau munculnya pesan eror, dan akar masalah (*root cause*), yaitu komponen yang

pertama kali rusak dan memicu seluruh rangkaian gangguan. Karena keduanya muncul bersamaan di banyak layanan, pelacakan akar masalah bergantung pada jejak yang tertinggal dalam data operasional sistem.

Penyebaran kegagalan ini tercermin dalam data telemetri, baik pada distributed traces melalui durasi eksekusi yang memanjang dan rentetan eror yang saling berkaitan dalam satu alur permintaan, maupun pada data metrik melalui lonjakan jumlah panggilan per menit serta peningkatan penggunaan CPU dan memori akibat beban kerja yang berpindah ke layanan lain [22]. Gabungan sinyal dari kedua jenis data ini memungkinkan algoritma diagnostik menemukan layanan yang menjadi titik awal kerusakan.

2.1.2 Root Cause Analysis

Pemulihan sistem dari kegagalan beruntun bertumpu pada *Root Cause Analysis* (RCA). Evaluasi ini bertujuan menemukan pemicu utama gangguan untuk mempercepat waktu perbaikan dan mencegah masalah terulang [10]. Namun, pencarian manual rentan keliru karena teknisi kewalahan menghadapi tumpukan data pemantauan dari berbagai layanan yang saling terhubung [23]. Keterbatasan inilah yang memaksa industri beralih ke sistem pelacakan otomatis.

Kebutuhan tersebut mendorong penerapan metode analisis otomatis berbasis data (*data-driven*). Inovasi saat ini berfokus menggabungkan berbagai rekaman pemantauan untuk diolah menggunakan algoritma *machine learning* [10]. Meskipun terbukti canggih, adopsinya masih lambat karena banyak tim SRE (*Site Reliability Engineering*) tetap bertahan pada pengamatan manual yang memakan waktu [23]. Kesenjangan inilah yang melatarbelakangi penelitian ini untuk mengembangkan model pencarian akar masalah yang akurat namun tetap transparan.

2.1.3 Observabilitas

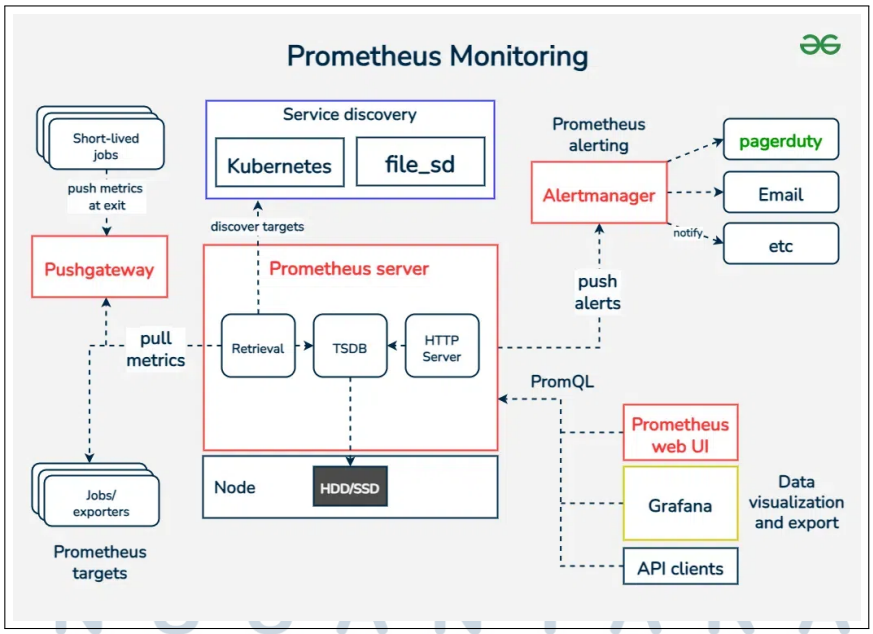
Observabilitas sistem *microservices* modern ditopang oleh tiga pilar utama, yaitu *log*, *traces*, dan metrik. Ketiga komponen ini memiliki peran operasional yang unik serta saling melengkapi, sehingga pemantauan menggunakan sumber data tunggal terbukti tidak lagi memadai untuk membedah arsitektur terdistribusi [23]. Untuk mengatasi titik buta pemantauan tersebut, penyelarasan berbagai bentuk data ke dalam satu analisis fusi telemetri sangat krusial guna merangkai pandangan

yang menyeluruh dalam mendeteksi anomali secara presisi [24]. Oleh karena itu, penelitian ini mengidentifikasi akar masalah dengan menggabungkan data metrik dan *distributed traces* secara bersamaan.

A Metrik

Metrik merupakan deret waktu numerik yang merekam kondisi operasional layanan secara kontinu. Data ini mencakup metrik sistem (*system metrics*) untuk memantau sumber daya fisik seperti CPU dan memori, serta metrik aplikasi (*application metrics*) untuk mengukur performa transaksi seperti latensi dan tingkat kesalahan [23]. Penggabungan kedua jenis metrik tersebut memungkinkan algoritma untuk mendeteksi anomali dan melokalisasi sumber masalah secara presisi.

Pada penelitian ini, data metrik bersumber dari dataset RCAEval yang dikumpulkan melalui kombinasi Prometheus, cAdvisor, dan Istio [6]. Dalam arsitektur tersebut, cAdvisor bertugas menangkap metrik penggunaan perangkat keras pada tingkat kontainer, Istio merekam metrik komunikasi jaringan antarlayanan di dalam *service mesh*, dan Prometheus bertindak sebagai pusat pengumpul seluruh data tersebut.



Gambar 2.1. Ilustrasi alur arsitektur Prometheus
Sumber: [25]

Prometheus mengumpulkan data melalui komponen *Retrieval* dengan

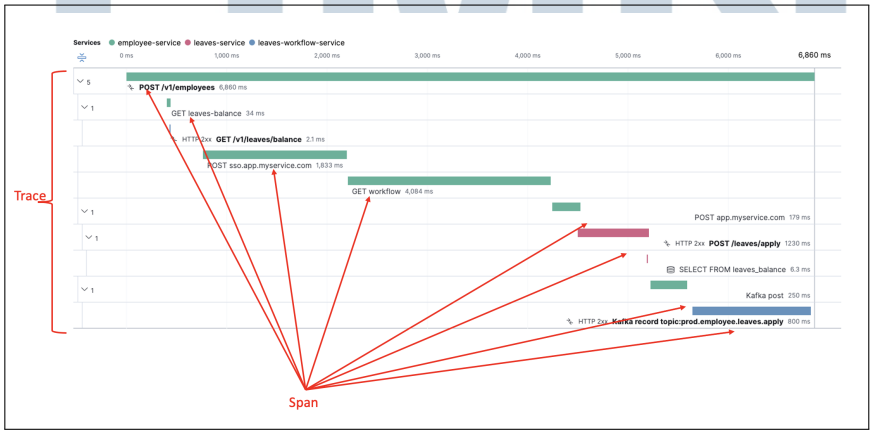
mekanisme penarikan (*pull metrics*) secara langsung dari berbagai target layanan atau *exporters*. Seperti yang ditunjukkan pada Gambar 2.1, data hasil penarikan ini disimpan ke dalam *Time Series Database* (TSDB) internal dan dapat diakses menggunakan kueri PromQL. Setiap entri data yang tersimpan memuat nama metrik, label dimensi (*key-value pairs*), nilai numerik, serta stempel waktu, dengan format dasar berikut:

```
<nama_metrik>{<label_key>="<label_value>", ...} <nilai> <timestamp>
```

Format pencatatan inilah yang menghasilkan kumpulan deret waktu multidimensi untuk mendokumentasikan fluktuasi kondisi setiap layanan, yang selanjutnya akan diolah menjadi masukan utama pada tahap ekstraksi fitur delta dalam model klasifikasi penelitian ini.

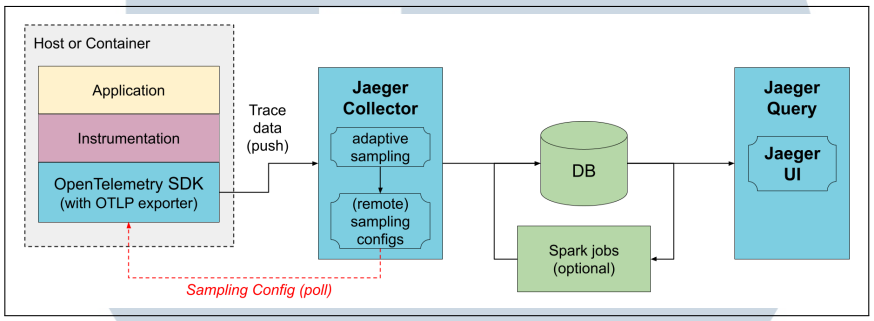
B Distributed Traces

Pelacakan terdistribusi memantau aliran permintaan secara menyeluruh melintasi berbagai layanan [23]. Setiap langkah operasi dicatat sebagai satu unit data yang disebut *span*. Unit ini menyimpan informasi penting seperti identitas layanan, penanda pelacakan (*trace ID*), penanda operasi (*span ID*), penanda operasi induk (*parent span ID*), waktu pencatatan, dan durasi [26]. Melalui penanda operasi induk tersebut, sistem dapat merangkai ulang urutan kejadian menjadi sebuah struktur pohon yang biasanya digambarkan dalam bentuk diagram garis waktu, seperti pada Gambar 2.2.



Gambar 2.2. Diagram garis waktu yang menampilkan hierarki operasi (*span*) dalam satu pelacakan.

Pada dataset RCAEval, perekaman ini dilakukan dengan memasang perangkat lunak OpenTelemetry pada setiap layanan. Perangkat lunak ini membuat penanda pelacakan yang unik dan meneruskannya ke layanan lanjutan melalui jalur HTTP (*Hypertext Transfer Protocol*) [6]. Mengacu pada Gambar 2.3, data operasi yang dihasilkan kemudian dikirim ke Jaeger Collector, diseleksi secara otomatis agar tidak membebani sistem, dan disimpan secara permanen di dalam Elasticsearch.



Gambar 2.3. Arsitektur pengumpulan data pelacakan menggunakan Jaeger.

Hasil akhir pengumpulan ini berupa sekumpulan data mentah dalam bentuk tabel. Seluruh informasi di dalamnya akan digunakan untuk memetakan hubungan ketergantungan antarlayanan sekaligus mencari letak pasti dari titik hambatan di dalam sistem.

2.2 Rekayasa Fitur Data Telemetri

Dalam literatur sistem terdistribusi, rekayasa fitur observabilitas dipahami sebagai proses konseptual untuk menjembatani data pemantauan *microservices* mentah agar dapat dicerna oleh algoritma pembelajaran mesin. Berbagai studi menetapkan bahwa model klasifikasi berbasis pohon seperti XGBoost memerlukan struktur data masukan yang konsisten agar dapat memetakan pola dengan andal [27]. Oleh karena itu, kerangka teoretis dalam penelitian terkait umumnya mencakup pemanfaatan pendekatan seperti analisis perubahan perilaku sistem, perangkuman data penelusuran, serta penyaringan atribut data untuk mendukung efektivitas identifikasi akar masalah.

2.2.1 Fitur Delta Temporal

Perbandingan jendela waktu sebelum dan sesudah kejadian anomali dapat dipahami sebagai representasi dari kegagalan sistem yang mengubah mekanisme generatif metrik pada layanan yang terdampak [28]. Data metrik yang terkumpul sebelum titik kegagalan membentuk kondisi operasional normal, sedangkan data yang terkumpul setelah titik tersebut mencerminkan kondisi anomali akibat perubahan mekanisme tersebut. Pemisahan aliran data telemetri ke dalam dua jendela berpasangan, yaitu jendela normal (*before*) dan jendela anomal (*after*), memungkinkan pergeseran perilaku sistem diukur secara eksplisit tanpa perlu memproses keseluruhan deret waktu yang bersifat kontinu dan tidak terbatas.

Persentil median (P_{50}) merepresentasikan perilaku operasional sistem pada kondisi umum, sedangkan P_{95} dan P_{99} menangkap *tail latency* yang tidak tercermin pada nilai median [29]. Dalam arsitektur dengan tingkat paralelisme tinggi, satu permintaan pengguna tidak selesai sampai sub-permintaan yang paling lambat juga selesai, sehingga latensi ekor pada segelintir permintaan dapat memengaruhi keseluruhan pengalaman pengguna. Ketiga persentil ini dihitung secara terpisah pada jendela normal dan jendela anomal untuk setiap kolom metrik telemetri.

Selisih antara persentil pada jendela anomal dan jendela normal dihitung sebagai fitur delta, sebagaimana dirumuskan pada Persamaan 2.1.

$$\Delta P_X = P_{X,\text{anomal}} - P_{X,\text{normal}} \tag{2.1}$$

Notasi X merujuk pada ketiga persentil yang dihitung, yaitu P_{50}, P_{95}, dan P_{99}. Kegagalan yang mengubah mekanisme generatif metrik secara menyeluruh [28], seperti pemadaman layanan, menggeser ketiga persentil secara bersamaan, sedangkan gangguan performa yang hanya memengaruhi sebagian kecil permintaan lebih terlihat pada ΔP_{95} dan ΔP_{99} sementara ΔP_{50} relatif stabil. Kombinasi ketiga fitur delta ini memberi model informasi soal besar sekaligus pola pergeseran nilai. Nilai delta ini menjadi representasi eksplisit dari pergeseran perilaku layanan akibat kegagalan sistem, dan diterapkan pada seluruh kolom metrik dataset RE2-OB dan RE2-TT sebagai vektor fitur masukan model XGBoost untuk mengidentifikasi layanan *root cause* pada arsitektur *microservices*.

2.2.2 Agregasi Fitur Traces

Atribut hasil agregasi traces dapat mengindikasikan komponen yang bermasalah, karena keberhasilan, kegagalan, durasi, dan frekuensi transaksi suatu layanan dapat dikorelasikan dengan kemungkinan layanan tersebut mengalami malfungsi [10]. Sistem seperti Pinpoint dan MEPFL memanfaatkan pola keberhasilan atau kegagalan permintaan pada tiap komponen untuk melokalisasi sumber kegagalan secara otomatis, sementara fluktuasi metrik transaksi seperti frekuensi pemanggilan turut merepresentasikan deviasi dari perilaku normal sistem. Atas dasar ini, tiga jenis fitur dihitung untuk setiap layanan, yaitu persentil latensi (P_{50}, P_{95}, P_{99}), frekuensi pemanggilan (`span_count`), dan tingkat kesalahan (`error_rate`).

Tingkat kesalahan dihitung sebagai proporsi span dengan kode status yang tidak sama dengan nol terhadap keseluruhan span pada suatu layanan, sebagaimana dirumuskan pada Persamaan 2.2.

$$\text{error\_rate} = \frac{\text{jumlah span dengan } \texttt{statusCode} \neq 0}{\texttt{span\_count}} \tag{2.2}$$

2.2.3 Penyaringan Fitur Varians Nol

Kutukan dimensi (*curse of dimensionality*) merujuk pada penurunan performa model pembelajaran mesin ketika jumlah fitur bertambah tanpa diimbangi jumlah data yang memadai, sehingga model rentan terhadap *overfitting* dan mahal secara komputasi. Seleksi fitur digunakan untuk menyederhanakan interpretasi model, mempersingkat waktu eksekusi, menghindari kutukan dimensi, serta meningkatkan generalisasi [30]. Konsep ini relevan bagi penelitian karena ekstraksi fitur delta dan agregasi *traces* menghasilkan data berdimensi tinggi yang perlu disaring sebelum dimasukkan ke model XGBoost.

Fitur varians nol adalah fitur bernilai konstan pada seluruh baris data, sehingga tidak membawa variasi yang dapat dipelajari model. Penyaringan fitur ini perlu dilakukan secara eksklusif pada set pelatihan, karena generalisasi model diukur dari kemampuannya memprediksi data yang belum pernah dilihat [27]. Jika penyaringan turut memanfaatkan set pengujian, terjadi kebocoran data (*data leakage*) yang membuat evaluasi performa model tidak lagi objektif.

2.3 Gradient Boosted Decision Tree

Transformasi data telemetri menjadi matriks fitur berdimensi rendah membutuhkan algoritma yang mampu menangkap pola nonlinier secara efisien. Keluarga algoritma ansambel berbasis pohon menawarkan pendekatan matematis yang tangguh untuk memetakan metrik operasional menjadi keputusan pelacakan masalah. Pembahasan ini menelusuri evolusi teoretis mulai dari mekanisme pemisahan dasar hingga optimasi tingkat lanjut pada kerangka kerja mutakhir.

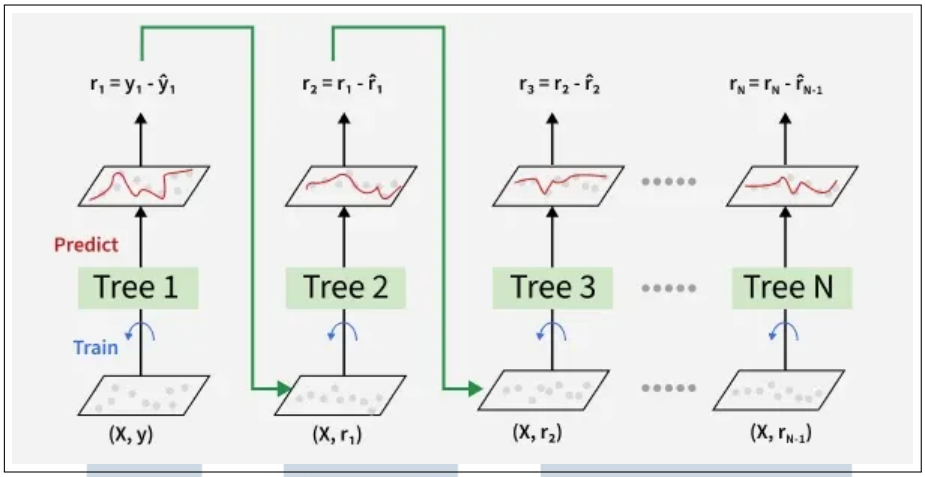
2.3.1 Pohon Klasifikasi dan Regresi

Pohon Klasifikasi dan Regresi (*Classification and Regression Tree* atau CART) merupakan algoritma pembelajaran mesin berbasis pohon keputusan yang membagi data secara rekursif menjadi kelompok-kelompok yang seragam. Proses klasifikasi bermula pada simpul akar dari pohon keputusan dan bergerak secara rekursif hingga mencapai simpul daun yang memuat label kelas [31]. Pada setiap simpul, algoritma menentukan fitur dan nilai pemisah yang paling optimal untuk menghasilkan kelompok data dengan label kelas yang seragam, dan proses pemisahan ini terus berlanjut hingga mencapai simpul daun yang memuat label kelas akhir.

Meskipun CART mampu memodelkan pola nonlinear dalam data, pohon keputusan tunggal rentan terhadap *overfitting* karena cenderung menghafal pola pada data pelatihan tanpa mampu menggeneralisasi dengan baik pada data baru [31]. Untuk mengatasi kelemahan ini, berbagai metode ansambel berbasis pohon dikembangkan dengan menggabungkan banyak pohon keputusan untuk menghasilkan prediksi yang lebih akurat, stabil, dan memiliki kemampuan generalisasi yang lebih baik, salah satunya melalui kerangka kerja *gradient boosting* (peningkatan gradien).

2.3.2 Gradient Boosting

Gradient boosting membangun model prediktif secara bertahap dengan menggabungkan sejumlah model lemah secara berurutan [32]. Setiap model baru dilatih khusus untuk memperbaiki kesalahan dari gabungan model sebelumnya, sehingga setiap iterasi mengarahkan model ke arah yang meminimalkan nilai fungsi kerugian secara bertahap.



Gambar 2.4. Ilustrasi alur kerja *gradient boosting*
Sumber: [33]

Sebagaimana diilustrasikan pada Gambar 2.4, pohon pertama dilatih pada data asli (X, y) dan menghasilkan prediksi awal. Pada setiap iterasi m, algoritma mencari fungsi pembelajar baru h_m beserta bobot optimalnya ρ_m yang meminimalkan total fungsi kerugian sebagaimana dirumuskan pada Persamaan 2.3. Optimasi ini sulit diselesaikan secara langsung sehingga digunakan pendekatan *gradient descent*.

$$(\rho_m, h_m(x)) = \underset{\rho, h}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \rho h(x_i)) \tag{2.3}$$

Pendekatan *gradient descent* dilakukan dengan menghitung *pseudo-residu* r_{mi} pada setiap iterasi yang dirumuskan pada Persamaan 2.4.

$$r_{mi} = \left[\frac{\partial L(y_i, F(x))}{\partial F(x)} \right]_{F(x)=F_{m-1}(x)} \tag{2.4}$$

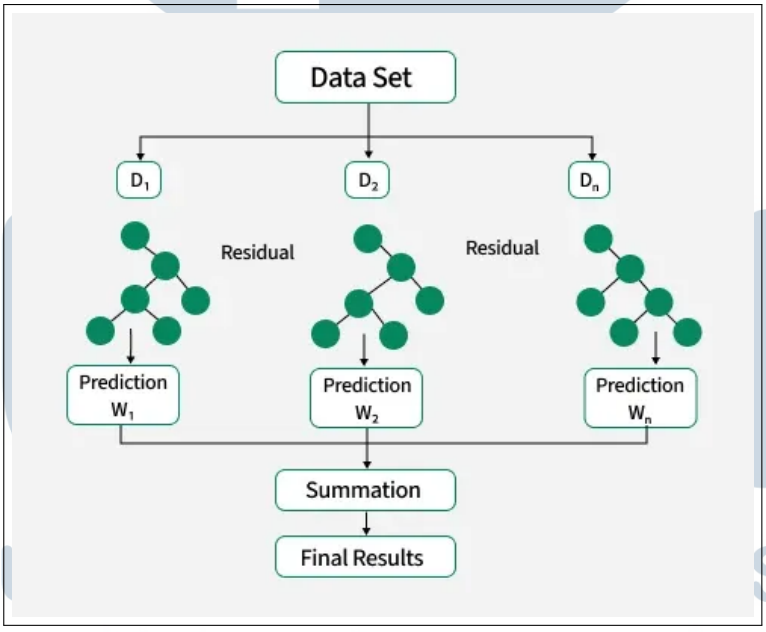
Nilai r_{mi} merupakan turunan parsial fungsi kerugian L terhadap prediksi model saat ini $F_{m-1}(x)$ yang mengukur arah dan besaran koreksi yang dibutuhkan. Pohon baru kemudian dilatih untuk memprediksi nilai residu tersebut, dan prediksi model diperbarui dengan menambahkan hasil koreksi dikalikan bobot ρ_m seperti pada Persamaan 2.5.

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x) \tag{2.5}$$

Proses ini diulang hingga N pohon terbentuk, menghasilkan *ensemble* yang secara kumulatif memetakan pola kompleks dalam data. Namun, kerangka kerja dasar ini tidak memiliki mekanisme pengendalian kompleksitas model sehingga berpotensi mengalami *overfitting*, keterbatasan yang kemudian diatasi oleh XGBoost melalui integrasi regularisasi struktural ke dalam proses optimasi.

2.3.3 Extreme Gradient Boosting

Algoritma XGBoost (*Extreme Gradient Boosting*) menyempurnakan kerangka kerja peningkatan gradien dengan mengintegrasikan regularisasi struktural, aproksimasi orde kedua, dan efisiensi komputasi ke dalam satu sistem [34]. Kerangka kerja utama dari algoritma ini dapat dilihat dengan jelas pada Gambar 2.5. Model XGBoost beroperasi sebagai ansambel dari pohon keputusan yang mengakumulasi kontribusi seluruh pohon secara aditif untuk menangkap pola nonlinier yang kompleks. Proses akumulasi prediksi dari berbagai pohon tersebut diformulasikan secara matematis melalui Persamaan 2.6.



Gambar 2.5. Kerangka kerja XGBoost (*Extreme Gradient Boosting*)
Sumber: [35]

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathscr{F} \tag{2.6}$$

Formula aditif tersebut memperlihatkan bagaimana model mengombinasikan beberapa komponen penting untuk menghasilkan prediksi akhir, yang meliputi:

1. K: Jumlah total pohon dalam ansambel.
2. $f_k(x_i)$: Prediksi dari pohon ke-k untuk sampel data x_i.
3. $\mathscr{F}$: Ruang fungsi CART, dengan definisi matematis $\mathscr{F} = \{f(x) = w_{q(x)}\}$.
4. q: Struktur pohon yang memetakan suatu sampel ke indeks daun yang bersesuaian.
5. w: Bobot atau skor kontinu yang terdapat pada daun tersebut.

Pohon regresi dalam XGBoost memiliki perbedaan mendasar dengan pohon keputusan tradisional karena menyimpan skor kontinu pada setiap daunnya alih-alih kelas kategorikal. Prediksi akhirnya murni didapatkan dari penjumlahan seluruh skor daun tersebut dari masing-masing pohon secara independen [34]. Pemetaan struktur dan bobot yang mendetail ini menjadi pembeda utama cara kerja internal XGBoost dibandingkan algoritma ansambel lainnya.

A Fungsi Objektif

Berbeda dari kerangka kerja peningkatan gradien dasar, XGBoost meminimalkan fungsi objektif yang secara eksplisit menyertakan faktor regularisasi untuk mengendalikan kompleksitas model [34]. Fungsi objektif ini memadukan dua komponen utama, yaitu fungsi kerugian untuk mengukur kesalahan prediksi dan nilai regularisasi untuk mencegah pembentukan arsitektur pohon yang terlalu rumit. Pada penelitian ini, klasifikasi multikelas memanfaatkan objektif `multi:softmax` dengan C sebagai representasi jumlah kelas layanan target. Oleh karena itu, fungsi kerugian dihitung menggunakan metode *cross-entropy loss* berdasarkan nilai probabilitas dari fungsi *softmax*. Seluruh komponen optimasi dan pembatasan model tersebut dirangkum secara utuh dalam Persamaan 2.7.

$$\mathscr{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{2.7}$$

$$l(\hat{y}_i, y_i) = -\sum_{c=1}^{C} \mathbf{1}[y_i = c] \log p_{ic}$$

$$p_{ic} = \frac{e^{\hat{y}_{ic}}}{\sum_{j=1}^{C} e^{\hat{y}_{ij}}}$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Formula objektif di atas memperlihatkan mekanisme model dalam menggabungkan optimasi prediksi dan pembatasan kompleksitas dengan rincian variabel sebagai berikut:

1. $\mathscr{L}(\phi)$: Fungsi objektif total yang diminimalkan oleh model.
2. $l(\hat{y}_i, y_i)$: Fungsi kerugian (*cross-entropy*) antara prediksi ($\hat{y}_i$) dan label aktual (y_i) untuk sampel ke-i.
3. $\Omega(f_k)$: Nilai regularisasi untuk pohon ke-k.
4. C: Jumlah total kelas target (jenis kegagalan atau layanan).
5. $\mathbf{1}[y_i = c]$: Fungsi indikator yang bernilai 1 jika sampel i masuk ke kelas aktual c, dan 0 jika tidak.
6. p_{ic}: Probabilitas prediksi bahwa sampel ke-i masuk ke kelas c.
7. $\hat{y}_{ic}$: Skor kontinu hasil akumulasi seluruh pohon untuk sampel i pada kelas target c.
8. γ: Parameter regularisasi untuk membatasi jumlah daun (T).
9. λ: Parameter regularisasi L2 terhadap bobot daun (w).

Fungsi indikator pada penjabaran tersebut memastikan bahwa perhitungan logaritma probabilitas hanya difokuskan pada kelas aktual yang bersesuaian. Faktor regularisasinya beroperasi secara ganda melalui pembatasan pertumbuhan daun serta penghalusan bobot untuk menekan risiko *overfitting* [34]. Keseimbangan

ini mencegah *overfitting* sehingga model tetap mampu menggeneralisasi data yang belum pernah dilihat sebelumnya.

B Optimasi Berbasis Gradien Orde Kedua

Optimasi langsung terhadap fungsi objektif sering kali sulit dilakukan karena tidak semua bentuk fungsi kerugian memiliki penyelesaian tertutup [34]. Untuk mengatasi keterbatasan tersebut, XGBoost menggunakan aproksimasi Taylor orde kedua untuk mengaproksimasi fungsi kerugian menjadi bentuk kuadrat pada setiap iterasi, sehingga bobot daun optimal dapat dihitung secara analitis dalam satu langkah. Pemanfaatan turunan orde kedua ini membuat proses pencarian nilai optimal menjadi lebih cepat dan stabil dibandingkan metode dasar sebelumnya. Setelah mengabaikan nilai konstan yang tidak bergantung pada pohon baru, fungsi objektif yang disederhanakan tersebut diformulasikan secara utuh melalui Persamaan 2.8.

$$\tilde{\mathscr{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{2.8}$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

Persamaan di atas menunjukkan bahwa optimasi pohon baru murni bergantung pada statistik gradien dari fungsi kerugian dengan rincian variabel sebagai berikut:

1. n: Jumlah total sampel data pelatihan.
2. $f_t(x_i)$: Prediksi dari pohon keputusan baru yang ditambahkan pada iterasi ke-t untuk sampel x_i.
3. g_i: Statistik gradien orde pertama dari fungsi kerugian (l) terhadap prediksi sebelumnya ($\hat{y}^{(t-1)}$).
4. h_i: Statistik gradien orde kedua atau Hessian dari fungsi kerugian terhadap prediksi sebelumnya.

5. $\Omega(f_t)$: Faktor regularisasi untuk mengendalikan kompleksitas pohon baru yang ditambahkan.

Statistik turunan pertama pada penjabaran tersebut mewakili arah kemiringan kesalahan dari prediksi yang dihasilkan sebelumnya. Keberadaan statistik turunan kedua memberikan informasi tambahan mengenai kelengkungan fungsi kerugian untuk memandu proses pencarian nilai optimal secara presisi. Penambahan faktor regularisasi pada bagian akhir persamaan berfungsi untuk memastikan bahwa setiap pohon baru yang terbentuk tidak menurunkan kemampuan generalisasi model [34].

C Konstruksi Pohon

Proses konstruksi pohon pada XGBoost menggunakan algoritma *greedy* untuk menemukan struktur yang meminimalkan fungsi objektif pada setiap iterasi [34]. Untuk suatu struktur pohon yang tetap, bobot optimal pada setiap daun dapat dihitung secara analitis seperti yang ditunjukkan pada Persamaan 2.9. Bobot ini mewakili kontribusi prediktif dari seluruh sampel di daun tersebut untuk mendongkrak performa model. Nilai bobot tersebut murni didapatkan dari rasio akumulasi gradien dengan penambahan parameter regularisasi untuk mencegah kemunculan prediksi yang ekstrem.

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{2.9}$$

Bobot optimal ini kemudian dievaluasi kualitasnya pada setiap kandidat pemisahan simpul cabang menggunakan skor *Gain*. Skor pemisahan tersebut menghitung seberapa besar reduksi kerugian dari fungsi objektif setelah suatu simpul dibagi menjadi dua anak simpul. Pemisahan murni hanya akan diaplikasikan apabila perhitungan tersebut bernilai positif atau ketika perolehan informasi melampaui batasan kompleksitas yang ditetapkan. Mekanisme pemangkasan otomatis ini mengendalikan kedalaman pohon tanpa memerlukan tahap pasca-pelatihan, di mana seluruh dinamikanya dirangkum dalam Persamaan 2.10 [34].

$$\mathscr{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma \tag{2.10}$$

Kedua persamaan di atas saling berkaitan untuk membentuk keseimbangan antara kemampuan prediktif dan kompleksitas struktural pohon dengan memanfaatkan berbagai variabel pembentuk yang meliputi:

1. w_j^*: Bobot atau skor kontinu optimal untuk daun ke-j.
2. $\mathscr{L}_{split}$: Skor *Gain* atau reduksi kerugian dari sebuah kandidat pemisahan simpul.
3. I_j: Himpunan indeks sampel data yang didistribusikan pada daun ke-j.
4. I_L, I_R: Himpunan indeks sampel pada simpul anak kiri dan anak kanan setelah pemisahan dilakukan.
5. I: Himpunan indeks sampel pada simpul induk sebelum pemisahan ($I = I_L \cup I_R$).
6. g_i, h_i: Statistik gradien orde pertama dan orde kedua (Hessian) dari fungsi kerugian untuk sampel ke-i.
7. λ: Parameter regularisasi L2 pada bobot daun.
8. γ: Batasan kompleksitas yang mendefinisikan ambang minimum *Gain* untuk kelayakan pemisahan simpul.

Parameter regularisasi pada susunan variabel tersebut berfungsi sebagai penahan agar bobot daun tidak tumbuh secara tidak terkendali. Batasan kompleksitas juga bertindak sebagai standar minimum yang sangat ketat untuk menentukan kelayakan eksekusi percabangan. Kombinasi kedua penalti ini secara otomatis membatasi model agar tidak sekadar menghafal noise dari data pelatihan. Interaksi yang seimbang dari seluruh komponen ini memastikan model dapat terus belajar meminimalkan kesalahan tanpa mengorbankan stabilitas generalisasinya [34].

D Klasifikasi Multikelas

Untuk menangani masalah klasifikasi multikelas, XGBoost memperluas kerangka kerjanya dengan menggunakan fungsi objektif *softmax* yang menghasilkan distribusi probabilitas di atas seluruh kelas target. Pada setiap iterasi, XGBoost membangun satu pohon per kelas secara paralel, sehingga untuk C kelas target dan K iterasi, total pohon yang dibangun adalah $K \times C$. Prediksi akhir

ditentukan sebagai kelas dengan probabilitas *softmax* tertinggi, yang dalam konteks penelitian ini berarti satu dari lima layanan target diprediksi sebagai *root cause* dari gangguan sistem tanpa memerlukan dekomposisi *one-vs-rest* [15, 16, 34].

2.4 Pelatihan dan Optimasi Model

Pelatihan dan optimasi model dilakukan untuk memastikan algoritma klasifikasi yang dibangun mampu menentukan akar masalah secara akurat, sekaligus mampu menggeneralisasi pada skenario kegagalan yang belum pernah ditemui sebelumnya. Validasi silang umum digunakan untuk mengestimasi performa model secara tidak bias sebelum dievaluasi pada data uji yang sesungguhnya [36]. Pada dataset dengan observasi berkelompok seperti pengulangan eksperimen dari skenario yang sama, validasi silang juga perlu diarahkan untuk mencegah kebocoran informasi antarkelompok. Sejalan dengan karakteristik tersebut, penelitian ini menerapkan validasi silang berbasis kelompok dan stratifikasi kelas, dipadukan dengan optimasi hiperparameter berbasis Optimasi Bayesian menggunakan Optuna.

2.4.1 Validasi Silang *Stratified Group K-Fold*

Validasi silang K-Fold adalah prosedur pengambilan sampel ulang yang mempartisi dataset menjadi K himpunan bagian yang saling independen [36]. Model dilatih pada $K-1$ lipatan dan diuji pada satu lipatan yang tersisa secara bergantian, kemudian kesalahan prediksi dari setiap lipatan dirata-ratakan menjadi estimasi performa akhir sebagaimana dirumuskan pada Persamaan 2.11.

$$Err \leftarrow \frac{1}{K}\sum_{k=1}^{K} Err_k \tag{2.11}$$

Dengan K menyatakan jumlah lipatan dan Err_k menyatakan kesalahan prediksi pada lipatan ke-k, nilai Err yang dihasilkan merepresentasikan performa model secara keseluruhan dan berfungsi sebagai alat diagnostik untuk mencegah *overfitting* [37].

K-Fold standar mengasumsikan setiap observasi bersifat independen, namun asumsi ini tidak terpenuhi pada dataset penelitian ini karena setiap skenario kegagalan memiliki tiga pengulangan yang saling berkorelasi. Penelitian ini menggunakan *StratifiedGroupKFold* untuk mengatasi hal tersebut melalui dua

mekanisme utama. Mekanisme pertama adalah *grouping*, yang memastikan seluruh pengulangan dari satu skenario selalu berada pada lipatan yang sama demi mencegah kebocoran informasi. Mekanisme kedua adalah stratifikasi, yang berfungsi menjaga proporsi setiap kelas target agar tetap merata di setiap lipatan.

2.4.2 Optimasi Bayesian TPE

Pencarian konfigurasi hiperparameter tidak dapat dilakukan melalui komputasi paksa akibat luasnya ruang pencarian pada algoritma ansambel tingkat lanjut. Algoritma *Tree-Structured Parzen Estimator* atau TPE menghadirkan kerangka kerja Optimasi Bayesian yang sangat efisien untuk menyelesaikan masalah tersebut. Berbeda dengan pendekatan proses Gaussian tradisional yang memodelkan probabilitas prediksi secara langsung algoritma ini mengambil rute probabilitas terbalik dengan memodelkan densitas $p(x|y)$ dan $p(y)$ [38].

$$p(x|y) = \begin{cases} l(x) & \text{jika } y < y^* \\ g(x) & \text{jika } y \geq y^* \end{cases} \tag{2.12}$$

Berdasarkan Persamaan 2.12 pendekatan ini membelah riwayat pengamatan menjadi dua distribusi kepadatan probabilitas [38]. Fungsi densitas $l(x)$ dibentuk dengan menggunakan observasi konfigurasi hiperparameter yang menghasilkan kerugian di bawah ambang batas y^* yang menandakan area pencarian menjanjikan [39]. Sementara itu fungsi densitas $g(x)$ merepresentasikan kelompok konfigurasi yang tidak menjanjikan. Untuk mengeksplorasi ruang hiperparameter secara cerdas TPE menghitung Fungsi Akuisisi melalui fungsi *Expected Improvement* atau EI.

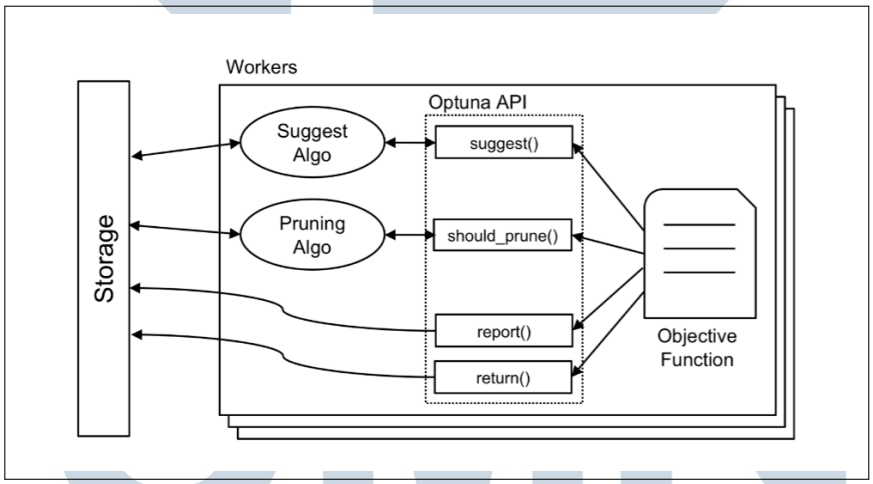
$$EI_{y^*}(x) \propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1} \tag{2.13}$$

Hubungan matematis pada Persamaan 2.13 menunjukkan bahwa optimalisasi perbaikan prediksi akan terjadi jika algoritma memilih titik konfigurasi yang memiliki probabilitas tinggi di bawah $l(x)$ dan probabilitas rendah di bawah $g(x)$ [38]. Struktur pembelahan probabilitas ini memungkinkan kerangka kerja TPE untuk menarik banyak kandidat konfigurasi baru secara independen dan mengevaluasinya dengan memaksimalkan rasio $l(x)$ terhadap $g(x)$ [39]. Logika pengambilan sampel secara independen ini sangat efisien secara komputasi dan

mampu menghasilkan konvergensi metrik yang luar biasa tajam sehingga kerangka kerja modern mengadopsinya sebagai mesin pencarian utama untuk menyetel parameter model.

2.4.3 Optuna

Optuna adalah kerangka kerja optimasi hiperparameter sumber terbuka yang dirancang untuk menyederhanakan dan mengotomasi proses pencarian konfigurasi model pembelajaran mesin [40]. Salah satu karakteristik utamanya adalah antarmuka *define-by-run*, di mana ruang pencarian hiperparameter dibangun secara dinamis setiap kali fungsi tujuan (*objective function*) dijalankan, berbeda dari kerangka kerja sebelumnya yang mengharuskan ruang pencarian didefinisikan secara statis di awal. Selain TPE yang telah dijelaskan pada Subbab sebelumnya, Optuna juga mendukung algoritma sampling lain seperti pencarian acak (*random search*), CMA-ES, dan optimasi Bayesian berbasis proses Gaussian.



Gambar 2.6. Arsitektur Optuna untuk optimasi hiperparameter
Sumber: [40]

Gambar 2.6 mengilustrasikan kerangka kerja Optuna. Setiap *worker* memanggil metode `suggest()` untuk memperoleh kombinasi hiperparameter baru dari algoritma sampling berdasarkan riwayat percobaan (*trial*) yang tersimpan pada objek penyimpanan (*storage*), kemudian melatih dan mengevaluasi model menggunakan kombinasi tersebut. Nilai evaluasi sementara dapat dilaporkan melalui metode `report()` dan diperiksa oleh `should_prune()` untuk menghentikan percobaan yang tidak menjanjikan, sehingga sumber daya komputasi dapat dialokasikan untuk kombinasi lain yang lebih berpotensi. Nilai akhir

dikembalikan melalui metode `return()` dan disimpan kembali ke *storage*, sehingga kombinasi hiperparameter yang diusulkan semakin terarah seiring bertambahnya jumlah percobaan, hingga konfigurasi dengan nilai evaluasi terbaik diperoleh sebagai hasil akhir.

Penelitian ini memanfaatkan Optuna dengan algoritma sampling TPE untuk menyetel enam hiperparameter XGBoost pada setiap kondisi eksperimen secara independen. Karakteristik *define-by-run* dan mekanisme penyimpanan riwayat percobaan pada Optuna memastikan setiap dari 100 percobaan yang dijalankan semakin terarah menuju konfigurasi hiperparameter yang optimal untuk masing-masing kondisi.

2.5 Metrik Evaluasi Model

Evaluasi kinerja algoritma klasifikasi pada ruang lingkup multikelas membutuhkan kerangka pengukuran yang holistik dan terbebas dari bias kelas mayoritas. Pemilihan metrik yang tepat sangat krusial untuk memastikan bahwa model benar-benar mempelajari pola anomali secara fundamental alih-alih sekadar menebak distribusi data dominan.

2.5.1 Confusion Matrix

Matriks kebingungan bertindak sebagai fondasi utama dalam mengevaluasi algoritma klasifikasi. Pada lingkungan operasional dengan m kelas matriks ini berukuran $m \times m$ di mana terdapat m klasifikasi yang benar dan $m^2 - m$ kemungkinan kesalahan klasifikasi [41]. Evaluasi prediksi dibangun di atas empat nilai kuadran dasar yaitu Positif Benar (TP) Negatif Benar (TN) Positif Palsu (FP) dan Negatif Palsu (FN). Seluruh turunan metrik evaluasi matematis pada dasarnya bertumpu pada probabilitas kombinasi keempat nilai kuadran tersebut.

2.5.2 Accuracy

Akurasi merupakan rasio pengukuran paling dasar yang menghitung proporsi keseluruhan prediksi yang berhasil ditebak dengan benar oleh model klasifikasi [42].

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.14}$$

Meskipun rumusan pada Persamaan 2.14 sangat intuitif metrik ini memiliki kelemahan fatal ketika dihadapkan pada himpunan data operasional yang timpang. Akurasi memberikan bobot rasio keberhasilan secara proporsional terhadap ukuran kelas sehingga secara sistematis mengabaikan performa model pada kelas minoritas. Berbagai simulasi membuktikan bahwa akurasi hanya berfokus pada keberhasilan klasifikasi mayoritas sehingga menghasilkan bias matematis tertinggi [43]. Sebuah model bahkan dapat mencapai akurasi sangat tinggi hanya dengan memprediksi kelas mayoritas secara terus-menerus meskipun sebenarnya model tersebut gagal total dalam mendeteksi anomali krusial.

2.5.3 Presisi dan Recall

Untuk mengatasi jebakan kelas dominan algoritma membutuhkan pengukuran yang berfokus langsung pada spesifisitas galat. *Recall* atau Sensitivitas mengukur kemampuan model dalam menangkap seluruh anomali yang sebenarnya terjadi dengan menghukum kesalahan model saat melewatkan kerusakan [41].

$$TPR = \frac{TP}{TP + FN} \tag{2.15}$$

Model yang terlalu sensitif dan agresif menebak anomali berpotensi menghasilkan banyak peringatan palsu. Oleh karena itu Presisi digunakan berdampingan dengan *Recall* untuk mengukur seberapa banyak tebakan anomali yang benar-benar relevan dengan menghukum penerimaan alarm palsu [41].

$$PPV = \frac{TP}{FP + TP} \tag{2.16}$$

Kedua elemen pada Persamaan 2.15 dan Persamaan 2.16 ini saling membatasi satu sama lain sehingga menuntut ekuilibrium matematis yang presisi.

2.5.4 F1-Score

F1-Score memformulasikan rata-rata harmonik antara Presisi dan *Recall* untuk memberikan satu metrik agregat yang seimbang dalam menilai ketepatan prediksi tanpa terjebak pada bias akurasi konvensional [41].

$$F\text{-}measure = 2 \times \frac{PPV \times TPR}{PPV + TPR} \tag{2.17}$$

Rumusan pada Persamaan 2.17 dihitung berdasarkan nilai Presisi dan *Recall* yang bersifat global atau per kelas tunggal, sehingga belum mencerminkan performa model secara keseluruhan pada kasus klasifikasi multikelas. Untuk mengatasi hal tersebut, diperlukan strategi pembobotan tambahan yang memperhitungkan performa pada setiap kelas secara individual.

2.5.5 Macro F1-Score

Macro F1-Score memperluas konsep *F1-Score* ke dalam konteks multikelas dengan menggabungkan rata-rata Presisi dan *Recall* per kelas, yaitu *Macro-Precision* dan *Macro-Recall*, sebagaimana dirumuskan pada Persamaan 2.18. *Macro-Recall* menghukum *omission errors* dengan mengevaluasi seberapa baik suatu kelas terdeteksi dari keseluruhan sampel aktualnya, sedangkan *Macro-Precision* menghukum *commission errors* dengan mengevaluasi relevansi suatu prediksi kelas terhadap label sebenarnya [44, 45]. Kedua metrik tersebut kemudian digabungkan melalui rata-rata harmonik untuk menghasilkan *Macro F1-Score*.

$$\begin{aligned} Macro\text{-}Recall &= \frac{1}{C}\sum_{j=1}^{C}\frac{TP_j}{TP_j + FN_j} \\ Macro\text{-}Precision &= \frac{1}{C}\sum_{j=1}^{C}\frac{TP_j}{TP_j + FP_j} \\ Macro\text{-}F1 &= 2 \times \frac{Macro\text{-}Precision \times Macro\text{-}Recall}{Macro\text{-}Precision + Macro\text{-}Recall} \end{aligned} \tag{2.18}$$

Karakteristik *macro-averaging* pada persamaan tersebut menjamin bahwa kelas anomali minoritas memiliki pengaruh pembobotan yang setara dengan kelas mayoritas, karena rata-rata dihitung tanpa mempertimbangkan jumlah sampel per kelas [45]. Karakteristik inilah yang menjadikan *Macro F1-Score* sebagai metrik optimasi yang dipilih pada penelitian ini, mengingat distribusi jumlah skenario antarlayanan target yang tidak sepenuhnya seimbang.

2.6 Explainable AI

Algoritma berbasis ansambel seperti XGBoost menghasilkan prediksi yang akurat namun sulit diinterpretasi karena kompleksitas struktur internalnya. Kerangka kerja *Explainable AI* (XAI) dirancang untuk menerjemahkan keputusan model menjadi penjelasan yang dapat dipahami, dengan mengalokasikan kontribusi setiap fitur terhadap prediksi secara kuantitatif [17].

2.6.1 SHAP

SHAP (*SHapley Additive exPlanations*) adalah metode atribusi fitur yang menerapkan konsep nilai Shapley dari teori permainan kooperatif untuk mengalokasikan kontribusi setiap fitur secara adil terhadap suatu prediksi [17]. Nilai Shapley ϕ_i untuk fitur ke-i dihitung dengan mengevaluasi kontribusi marjinal fitur tersebut terhadap seluruh kemungkinan kombinasi (*subset*) fitur lainnya, sebagaimana dirumuskan pada Persamaan 2.19.

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \left[v(S \cup \{i\}) - v(S) \right] \tag{2.19}$$

Dengan N menyatakan himpunan seluruh fitur, S menyatakan subset fitur tanpa fitur ke-i, $v(S)$ menyatakan prediksi model menggunakan hanya fitur dalam S, dan $v(S \cup \{i\})$ menyatakan prediksi model ketika fitur ke-i ditambahkan ke S. Persamaan 2.19 pada dasarnya merata-ratakan selisih prediksi $v(S \cup \{i\}) - v(S)$ pada seluruh kemungkinan urutan penambahan fitur, sehingga kontribusi setiap fitur dapat dihitung secara adil terlepas dari urutan evaluasinya.

SHAP kemudian mendefinisikan model penjelasan sebagai fungsi linier dari seluruh atribusi fitur ϕ_i tersebut, di mana penjumlahannya harus tepat sama dengan output prediksi model, sebagaimana dirumuskan pada Persamaan 2.20.

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{2.20}$$

Dengan M menyatakan jumlah fitur dan ϕ_0 menyatakan nilai dasar (*baseline*) prediksi model, properti ini disebut *local accuracy* dan menjamin bahwa penjelasan yang dihasilkan selalu konsisten dengan prediksi aktual model [17]. Nilai ϕ_i untuk setiap fitur diperoleh langsung dari kalkulasi TreeExplainer, dan digunakan dalam

penelitian ini untuk membandingkan besaran kontribusi fitur metrik dan fitur *trace* terhadap prediksi model XGBoost pada setiap arsitektur.

2.6.2 SHAP TreeExplainer

Komputasi nilai Shapley secara eksak bersifat NP-*hard* karena mengharuskan evaluasi atas seluruh kemungkinan kombinasi fitur. TreeExplainer mengatasi keterbatasan ini dengan algoritma *dynamic programming* yang menyusuri struktur pohon keputusan secara rekursif, sehingga nilai Shapley yang eksak dapat dihitung dalam waktu polinomial tanpa perlu mengevaluasi seluruh kombinasi fitur [18]. Selain penjelasan lokal per prediksi, TreeExplainer juga menghasilkan kepentingan fitur global dengan merata-ratakan nilai atribusi dari seluruh sampel dalam dataset, memberikan gambaran menyeluruh mengenai fitur mana yang paling berpengaruh terhadap keputusan model secara keseluruhan. Dalam penelitian ini, TreeExplainer diterapkan pada model XGBoost multimodal untuk menghasilkan kepentingan fitur global yang digunakan sebagai dasar analisis kontribusi modalitas telemetri pada Bab 4.